# A Performance Evaluation of a Grid-enabled Object-Oriented Parallel Outdoor Ray Launching for Wireless Network Coverage Prediction

Zhihua Lai*†, Nik Bessis*, Pierre Kuonen†, Guillaume de la Roche*, Jie Zhang* and Gordon Clapworthy*

*Institute for Research in Applicable Computing
University of Bedfordshire
Luton LU1 3JU - UK

†GRID and Ubiquitous Computing Group.
University of Applied Sciences of Fribourg
UCH-1705 Fribourg - Switzerland

{zhihua.lai, nik.bessis, guillaume.delaroche, jie.zhang, gordon.clapworthy}@beds.ac.uk, pierre.kuonen@eif.ch

***Abstract*—The use of parallel technologies to solve complex scientific problems has gained increased popularity. The ray optical methods are deterministic propagation approaches that are based on geometrically searching paths between emitter and receivers. They offer higher accuracy than empirical models, but suffer from slow calculations on exhausted rays that have to be searched. In this paper, an object-oriented scheme based on POP-C++ for parallel objects to accelerate outdoor ray launching is described. Performance evaluation is presented to show that this parallel scheme is promising in outdoor wireless propagation modeling. The possibility of running this model in the distributed grid environment is also discussed. Results have shown the great potential of using such a parallel model to predict accurate outdoor wireless propagation scenarios within a short time.**

***Index Terms*—Parallel ray launching, POPC++, radio propagation, outdoor coverage prediction, performance evaluation.**

## I. Introduction

During the last few years, wireless propagation modeling has been repeatedly quoted as a key factor in the planning and optimization process of a 3G/4G network [1][2]. Because of the rapid development of radio networks, there is an increasing need for fast and accurate 3D propagation models.

Ray Optical (RO) methods are deterministic approaches that are being frequently used. They are generally computation intense but give a high accuracy [3]. This can be generally divided into two categories: ray tracing and ray launching which can be distinguished by the way they handle ray searching [4][5]. RO are based on geometrically searching possible rays between emitter and receivers. Usually they are more suitable in the urban scenarios rather than indoor. This is because outdoor ray optical methods suffer less from material aspects since the given database usually is not complete (only contains few material types such as concrete for buildings, trees etc). In contrast, material effects play a much more important role in indoor modeling which limits the performance of ray optical methods [6]. In both cases, the performance of RO is usually restricted by a limited number of combined ray iterations. Parallelism of RO improves the performance which has been addressed in literature. For example, in [7], a parallel approach for 3D ray tracing has been proposed with some interesting results.

IRLA, namely Intelligent Ray Launching Algorithm is presented in [8], which gives a fast and accurate coverage prediction for 3D outdoor scenarios. IRLA utilizes ray launching for outdoors by fast calculation of roof-top diffractions which are considered as dominant rays for urban scenarios [8][4]. The combined diffractions and reflection for horizontal planes are also utilized.

The rest of the paper is organized as follows: in Section II, the basic structure of IRLA algorithm for outdoors is described, followed by the discussion of a parallel version of IRLA in Section III, with the potential to run IRLA in a distributed grid environment. Results are given in Section IV followed by Section V which concludes our current work.

## II. Wave propagation with IRLA

IRLA consists of three main components [8]: Light-Of-Sight engine (LOS), Vertical Diffraction (VD) engine, and Horizontal Diffraction and Reflection engine (HDR). The effect of antenna pattern can be included in LOS engine or they can be dealt in a post-processing to adjust the prediction values. In this section, the IRLA is introduced with the discussion about multi-threading improvement.

### A. *Single processor version*

IRLA relies on the cubic data obtained from discretization of the environment. One cubic element can represent building walls, ground, trees etc. IRLA collects LOS pixels (visible to emitter) by scanning cubes along pre-defined directions. To avoid the problem of missing pixels caused by the dispersion of rays with a constant angle defined, all discrete rays connecting emitter and the fringe pixels of the scenario are considered. The total rays can be determined. As the LOS does not have iterations of ray propagating, it can be finished within a time of $O(n^2)$ magnitude. The pixels obtain by LOS engine should be ready for HDR engine, which iterates the ray launching. HDR is responsible for finding reasonable ray paths of combined reflections and diffractions horizontally (Fig. 1).

VDR has the complexity of $O(n^3)$, which mathematically calculates the number of roof-top diffractions for a outdoor ray path. It launches the ray vertically and intelligently addresses the vertical diffractions by actively connecting paths between buildings. HDR is the most time-consuming part of IRLA. It launches a complete set of combinations of diffraction and reflection rays. The running time depends on the number of maximum iterations, the number of obstacles and a threshold signal strength used to terminate radio waves that fall below. Despite the cut-off searches of shadow area of diffraction [8], HDR still requires much computation power.

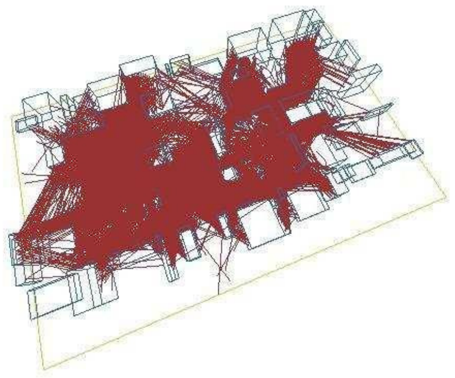


Fig. 1. HDR Reflections and Diffractions in Urban

Post-processing of IRLA can include antenna pattern adjustment and indoor prediction. If the indoor coverage is not required, the waves are therefore ignored when they penetrate the buildings. A considerable memory and computation power can thus be saved. On the other hand, an empirical loss can be added to indoor prediction.

## *B. Multi threading version*

Potentially, IRLA can run faster on multi-processor machines by simply implementing multi-threading. Usually the operating system can schedule more cycles on multi-threading than the normal single-threaded application. For example, the CPU can be better utilized (around 90%) rather than approximately 50% on a dual-core machine. However, the performance is limited because of physical resources (i.e. limited CPU, memory) and possible delays caused by thread synchronization. In Fig. 2, it is observed that with the few threads used, the performance will be increased but it quickly reaches to the limit and this performance tends to degrade as the number of threads increases afterwards. This is due to the fact that the race conditions of threads tend to happen when there are many threads. It is also noticed that multi-threading IRLA consumes memory as the number of threads grows, which in a way limits the performance acceleration.

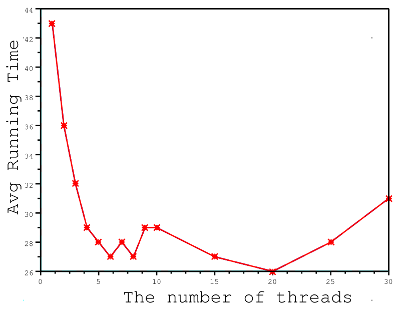


Fig. 2. The Performance of Multi-threaded IRLA

## III. Multi processor version of IRLA

In this section, the parallel scheme that applies to IRLA is given. The performance analysis method is closely related to Bulk Synchronous Programming (BSP) model[9]. BSP is used to design parallel algorithms which can be divided into concurrent computation, communication and barrier synchronization. IRLA is parallelized as described next. The future potential to run IRLA in distributed grid environment [10][11] is also discussed.

## *A. Parallel model*

IRLA is distributed to parallel objects. Each object handles parts of the parallel task and finally the results are collected. As IRLA is composed of three parts as mentioned in II, the parallel IRLA divides the computation, distributes them among processors and collects results, as given in Algorithm 1.

**Algorithm 1** Parallel IRLA for Outdoor

```
Parallel_IRLA(Master, Children, Tx)
  Master.Create(); Master.LoadData();
  for all $N$ in Children do
    $N$.Create();
    $N$.LoadData();
  end for
  Master.WaitForAll(Children);
  Master.IRLA_LOS_Fully();
  Children.IRLA_LOS_Partially();
  for all $N$ in Children do
    $N$.HDR();
  end for
  Master.WaitForResults(Children);
  for all $N$ in Children do
    $N$.VD();
  end for
  Master.WaitForResults(Children);
  Master.PostProcess();
```

The objects are created in parallel on the master node $M$ and children nodes $N_1$, $N_2$, ... $N_p$. On creation, they are given an ID. Building data, antenna data and network configurations have to be loaded by all objects before actual simulation starts. This is ensured by setting up a barrier. As the time of loading data can usually be trivial, the cost of this barrier can usually be neglected. Because LOS engine has a lower computation complexity compared to other components, it will only be performed fully on the node that the result is stored (in this case, on the master node), while the rest of the nodes would just simply obtain LOS pixels for the use of a HDR engine. This will avoid unnecessary communication overhead spent on trivial tasks.

HDR engine will be performed concurrently on all objects. Each object is responsible for parts of the computation, which is divided by the number of processors based on their unique identity number. Double Marking (DM), occurs when duplication of rays are mistakenly calculated [4], which will result in incorrect results and waste a lot of computation power. To handle DM as efficient as possible, the following rules apply.

- DM can be efficiently handled locally, instead of globally, among processors. This is due to the expensive overhead needed by communication and synchronization among parallel objects. To avoid this, it is required to distribute rays of a continuous region to each object. Therefore each processor will utilize local cache hit to avoid as much DM as possible.
- DM will occur often on the border of regions of rays assigned, in this case, DM can be avoided on the master node by ensuring the critical session that no two processor access the same pixels at the same time.

The VD Engine can be parallelized in a similar way. Each parallel object is responsible for parts of computation, which is a continuous region. At the master node, results are collected. Since the complexity of VD is constant, independent of the number of diffractions defined, the speed up of the VD engine by employing more processors has not been clearly observed. In contrast, by using more processors on the calculation of VD will incur overhead of communication. The efficiency of the parallel VD engine can be modeled by the following formula (which derives from [9]):

$$\text{COST}_{\text{VD}} = \max_{i=1}^{p}(T_i) + \max_{i=1}^{p}(M_i) + l \tag{1}$$

where $p$ represents the number of parallel objects;
$T_i$ is the time of local computation by process $i$;
$M_i$ is the time of global communication (messages sent and receiver by process $i$);
$l$ is the cost of setting up the barrier for synchronization.
If communication expense is higher than maximum computation time, there is no need to involve more processors on the VD engine. The model given here assumes homogeneous processors. Therefore, the computation power is treated equally for every parallel object. In reality, processors are usually have different capacities, which will be discussed in the next section.

To avoid as much communication as possible, the job distribution scheme is dynamically obtained on each object based on the total size of work and its object ID. To avoid using up too much memory space on children nodes, the temporary results obtained after each engine are sent to the master node. It has been observed that as many as possible results should be collected instead of sending pieces of data. Setting up communication can be quite costly, and thus, this should be prevented whenever possible.

The master node has to perform post-processing, which includes the antenna pattern adjustment and indoor points prediction. The indoor coverage prediction will usually be empirical by adding a empirical loss through building walls by considering the signal strength of outside buildings. This part of work is trivial compared to HDR and VD engine and hence there is no need for it to be parallelized.

## *B. Grid-enabled IRLA*

As a matter fact, the resources available in the grid are different in terms of memory and computation power [12]. The reason for running IRLA in the grid is to allow the identification of suitable resources, which potentially will make IRLA run faster and thus possibly be able to solve large problems.

POP-C++ [13] is a high level C++ object oriented programming language that is designed for distributed algorithms. By using POP-C++, it is flexible to use the job manager, which specifies the requirements of nodes (such as min memory and power) that jobs are run on rather than the exact machines. However, the runtime environment does not guarantee an optimal match of requirements. Rather, it searches the known nodes and returns the first one that satisfies the requirement given [13]. Assume the nodes searching time is trivial, the importance of finding optimal resources in the grid to be able to run IRLA can be clearly seen in:

- Finding nodes of similar power and memory that will minimize the waiting time of faster processors: all sub-jobs will be finished roughly at the same time.
- Identifying optimal distribution of jobs by the capacity of nodes available, which will also shorten the waiting time. For example, faster processors handle more and/or harder jobs.

However, in reality, the exact running time cannot be guaranteed. Therefore, only estimation can be analyzed. Assume there are $n$ ($N_1$, $N_2$ ... $N_n$) nodes available and $M$ nodes are required to run IRLA. Their costs of communication are empirically listed as $C_1$, $C_2$ .. $C_n$ and the computation power (in terms of M-flops), which are dynamically obtained as $P_1$, $P_2$ .. $P_n$. Note this value is adjusted during runtime (empirically) according to the usages of nodes.

Assume the tasks of IRLA can be noted as $J_1$, $J_2$ and $J_3$, for IRLA LOS, HDR, and VD respectively, the parallel model can be analyzed theoretically. Two rules can be observed: the bigger $P$ the better, and the smaller $C$ the better. Unfortunately there is no relation between $P$ and $C$, and hence a performance index (such as $F_i = uC_i \times wP_i$, $u$, $w$ are coefficients) can be defined. By sorting $F_i$ in non-ascending order, the candidate nodes to run IRLA can be chosen. By applying weighting function on $F$, (such as $j_i = \frac{KF_i}{\sum_{k=1}^{n} F_k}$, $K$ represents the computation intensity of the job) each node obtains the amount of jobs depending on the performance index. Generally, the bigger $F$ the more jobs a node gets. Unlike VD, and LOS, the simulation activity of HDR engine can not be fully predictable because of unforeseen ray phenomenal. For example, a ray

TABLE I
NETWORK CONFIGURATIONS OF MUNICH SCENARIO

| | |
|---|---|
| Area | 8.1 km$^2$ X 100m |
| Resolution | 5 X 5 X 5 |
| Maximum Reflection | **15** |
| Maximum Horizontal Diffraction | **15** |
| Maximum Vertical Diffraction | **Unlimited** * |
| Maximum Transmission | **Unlimited** * |

* until signal strength is under threshold

may be terminated early because its signal strength falls below a threshold value. VD requires the largest computation resources. By roughly dividing VD among objects (assume the use of homogeneous processors), the running time of each sub-jobs vary at runtime. Therefore, this will cause a cost in synchronization of faster processors. One possible solution to this can be through the empirically estimation for the density of urban sub-regions. This will be explored in further work.

## IV. RESULTS

In this section, the results of running parallel IRLA objects will be presented. COST-231 Munich Scenario [14] will be used as a benchmark. It has been shown in [8] that after a proper calibration of the materials, such model provides a *RMSE* (Root Mean Square Error) of $6dB$ between simulation and measurements. This high accuracy makes IRLA efficient for wireless network planning. However, in order to test a lot of parameters for the base stations, it is also important to reduce as much as possible the simulation time.

To fully evaluate the performance of parallel IRLA in term of speed, a complex network configuration has been chosen as shown in Table I (the number of combined reflections and diffractions in the horizontal plane is increased). Pre-defined empirical loss values for reflection, diffraction and transmission are given as 6, 8 and 15dB respectively, which yields a loss range of around 200 dB (a combination of 15 reflection and 15 diffraction). The results of parallelizing each component of IRLA are given below, followed by the description of final results with relevant performance issues analyzed.

### A. LOS

LOS is fully performed on the master node where pixels are obtained from (Fig. 3), the rest of the nodes run a partially LOS which purely obtain visible secondary pixels that are used in VD. The experiment has shown that the running time on LOS on a standard PC (Table II) is around 6 seconds.

### B. HDR

HDR is considered the most time-consuming part of IRLA. Since the complexity of LOS is trivial and of VD is constant, the overall acceleration of parallelism relies on HDR. That is to say, if the complexity of this part is low (e.g. few ray iterations specified), it would have run faster without parallelism due to the communication overhead that would occur.

No clear job distribution scheme can be easily observed, which is due to the rays bouncing around in dense urban environment. It also shows that the radio wave can hardly propagate far through reflection and diffraction in an horizontal plane (Fig. 4 shows the propagating effects of a high number of combinations of reflections and diffractions).

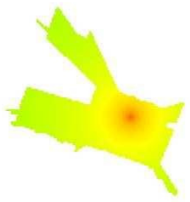
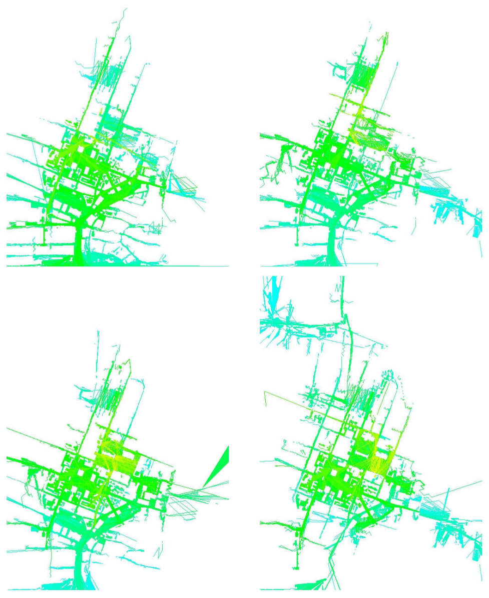


Fig. 3. The pixels obtained by master node

Fig. 4. HDR obtained by 4 different nodes (7 processors in all)



### C. VD

The most dominant rays for dense urban are roof-top diffractions which can be modeled by VD, which actively checks the paths of shortest vertical diffractions. VD has a low complexity, which depends linearly on the size of scenario, unlike HDR, the complexity of which is largely affected by the number of buildings. In Fig. 5, each processor has been assigned an equal piece of work. The job distribution can be easily observed, which is because VD only considers the shortest diffraction path over roof-tops without considering reflections.

### D. Final results

The data collected from children nodes are finally merged together, which results in the coverage prediction for Munich city as depicted in Fig. 6.

Fig. 5. VD obtained by 3 different nodes (7 processors in all)

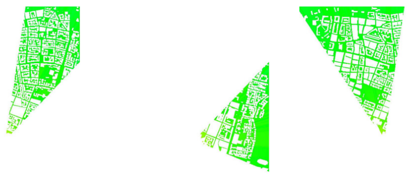


TABLE II
NODES SPECIFICATION

| | |
|---|---|
| CPU | 1.6GHz |
| Cores | 1 |
| RAM | 512MB |
| Software | Ubuntu 8.1, POP-C++ 1.2 |

The parallel simulation with different number of nodes have been carried out. By concurrently performing heavy computation, the simulation time (Table III) is shortened. It has been observed that the running time of VD is trivial so the cost of synchronization $T_{\text{Bvd}}$ is small. There is little space for this part of work to be speed up. On the other hand, HDR can be much shorten by utilizing more processors. LOS engine does not vary too much because the jobs are equally divided and can be determined beforehand. As the number of parallel objects is incremented, the cost for communication and synchronization tends to grow. Using parallelism, the performance is doubled by simply using 8 processors.

It is noticed that it is possible to create more than one object on the same node, which locally utilizes CPU usages. These objects are logically independent of each other, although they are created on the same physical machine. The cost to initialize them will be unnecessarily expensive, which limits the number of objects that can be created because the memory required is linearly dependent on the number of objects. This can be avoided by using multi-threading technology.

TABLE III
RUNNING TIME (SECONDS)

| $N$ | $T$ | $T_{\text{LOS}}$ | $T_{\text{HDR}}$ | $T_{\text{Bhdr}}$ | $T_{\text{VD}}$ | $T_{\text{Bvd}}$ |
|---|---|---|---|---|---|---|
| 1 | 132.0 | 6.5 | 39.7 | 9.0 | 9.4 | 0.2 |
| 2 | 101.1 | 6.3 | 33.3 | 14.8 | 5.7 | 0.2 |
| 3 | 80.2 | 6.3 | 32.5 | 23.5 | 4.4 | 0.1 |
| 4 | 71.0 | 6.6 | 27.1 | 21.9 | 4.4 | 0.2 |
| 5 | 69.9 | 6.3 | 26.9 | 23.4 | 3.2 | 0.1 |
| 6 | 66.4 | 6.7 | 26.7 | 26.3 | 3.2 | 0.2 |
| 7 | 64.1 | 6.6 | 25.8 | 22.0 | 2.5 | 0.1 |
| 8 | 63.9 | 6.2 | 23.4 | 22.0 | 2.6 | 0.2 |

where
$N$ represents the number of nodes used
(the specification is listed in Table II);
$T$ represents the total running time,
which includes the time of communication;
$T_{\text{LOS}}$ represents the maximum computation time for LOS engine;
$T_{\text{HDR}}$ represents the maximum computation time for HDR engine;
$T_{\text{VD}}$ represents the maximum computation time for VD engine;
$T_{\text{Bhdr}}$ represents the maximum object idle time
after the barrier of HDR engine;
$T_{\text{Bvd}}$ represents the maximum object idle time
after the barrier of VD engine;

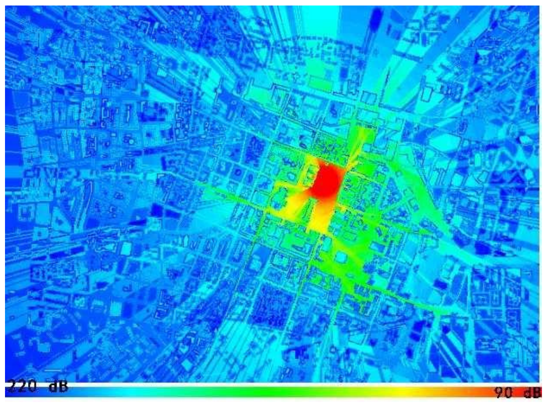


Fig. 6. The final results collected at master node

## V. CONCLUSION AND PERSPECTIVES

This paper has presented a parallel scheme to run IRLA (Intelligent Ray Launching Algorithm for Urban). The potential to employ grid technology is also discussed. Experiments have been carried out by using up to 8 parallel objects. It is observed that IRLA benefits from parallelism. An intelligent job distribution scheme remain undefined. Further work includes adding multi-threading support for parallel objects. It is believed this will improve the performance because the cost to start a thread is less than a process (object) [15].

The experiments have also shown the problem of equally dividing jobs among processors: faster processors would finish jobs early, thus they have to wait for each barrier synchronization. This is costly and has to be avoided in further works by designing and implementing intelligent job scheduling algorithms for POP-C++.

Parallel IRLA can potentially run faster by considering a small but enough number (instead of 15 reflections and 15 diffractions used for performance evaluation purposes) of ray iterations. It can be further accelerated by only calculating ground-level pixels for outdoors. Wireless network characteristics such as the effective antenna radiation can be studied in further work.

BSP model [9] has been shown to fit into IRLA for outdoor, and is expected to be applied to IRLA for indoor in the future.

The use of ”MapReduce” (”which is a software framework introduced by google to support distributed computing on large data sets on clusters of computer”)[16] is currently being investigated, which could be beneficial in improving the overall performance of parallel IRLA.

## ACKNOWLEDGEMENTS

This work is supported by the EU-FP6 ”RANPLAN-HEC” project under grant number MEST-CT-2005-020958, ”GAWIND” under grant number MTKD-CT-2006-042783 and

"Marie Curie Fellowship for Transfer of Knowledge". The following people deserve acknowledgements for their help on POP-C++ programming: Jean-Francois Roche and Laurent Winkler.

## References

[1] H. Holma and A. Toskala. *WCDMA for UMTS, Radio Access For 3G Mobile Communications Third Edition*. John Wiley & Sons Ltd, 2004.

[2] J. Oostveen and O. Mantel. Requirements on ray-based propagation models for mobile network planing, 6 2008. COST2100 Temporary Document (08)517.

[3] Y. Lostanlen. Radio wave propagation part two 'practical aspects'. In *1st COST 2100 Training School*, Wroclaw, Poland, 2 2008.

[4] R. Mathar, M. Reyer, and M. Schmeink. A cube oriented ray launching algorithm for 3d urban field strength prediction. In *IEEE Communications Society*, volume 49, pages 5034–5039, 6 2007.

[5] T. Kurner. Radio wave propagation part one 'theoretical aspects'. In *1st COST 2100 Training School*, Wroclaw, Poland, 2 2008.

[6] J. Jemai and T. Kurner. Towards a performance boundary in calibrating indoor ray tracing models. In *EURASIP Journal on Wireless Comm. and Net.,*, Feburary 2009.

[7] A.M. Cavalcante, M.J. de Sousa, J.A. Costa, C. Frances, and G.P. dos Santos Cavalcante. A parallel approach for 3d ray-tracing techniques in the radio propagation prediction. In *Journal of Microwaves and Optoelectronics*, 6 2007.

[8] Z. Lai, N. Bessis, G. de la Roche, H. Song, J. Zhang, and G. Clapworthy. An intelligent ray launching for urban propagation prediction. In *3rd European Conference on Antennas and Propagation*, Berlin, Germany, 3 2009.

[9] R.H. Bisseling. *Parallel Scientific Computation: A Structured Approach Using BSP and MPI*. Oxford University Press, 2004.

[10] Z. Lai, N. Bessis, J. Zhang, and G. Clapworthy. Some thoughts on adaptive grid-enabled optimisation algorithms for wireless network simulation and planning. In *UK e-Science, All Hands Meeting*, Nottingham, UK, September 2007.

[11] I. Foster and C. Kesselman. *The Grid2, blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers Inc., 2003.

[12] I. Foster. What is the grid? a three point checklist, 7 2002.

[13] T.A. Nguyen and P. Kuonen. Programming the grid with pop-c++. In *Future Generation Computer Systems*, HCMC University of Technology, Faculty of Computer Science and Engineering, Ho Chi Minh City, Viet Nam, 1 2007.

[14] COST 231 - urban micro cell measurements and building data. http://www2.ihe.uni-karlsruhe.de/forschung/cost231/cost231.en.html.

[15] A. Silberschatz and P.B. Galvin. *Operating System Concepts with Java (7th edition)*. John Wiley & Sons, Inc., 2006.

[16] J. Dean and S. Ghemawat. Simplified data processing on large clusters, 2004.